CS655 (Networks) Programming Assignment 0
Wil Koch, Sophia Yakoubov October 8, 2014

1 Program Documentation

We implemented this assignment in Python, and our architecture is built of off Simpy, a process-based discrete-event simulation library. Each source and the router is modelled as a processes while the transportation of packets are modelled as events.

1.1 Queuing Algorithms

In our architecture we utilize a process interaction concept in Simpy known as Stores which allows the modelling of production and consumption of objects. Stores allow objects to be put and get from processes. We abstract this concept to allow us to model the receiving and transmitting of packets by the router. When packets are sent to the router they are put in the Store, likewise when the router transmits a packet from a queue it is get from the Store. Fortunately Simpy allows for subclassing the Store class to provide your own implementation for how objects are put and get from the Store. We created three subclasses, for each queuing algorithms providing a robust flexible framework that allows different algorithms to be easily simulated. These classes are store_fifo.py, store_rr.py and store_drr.py.

1.2 Sources

Simpy provides another process interaction concept called Containers which allow the modelling of quantities of similar discrete objects. In order to limit the number of packets used in an experiment we use a container and "fill it" with the number of packets we want to run. Each source will then decrement the value in this container each time it creates a packet. When The container is empty, sources will no longer be able to create packets and once the router transmits the last packet the simulation will terminate.

Each packet created by a source is represented by a packet object specifying its sequence number, source/destination addresses and the length of the packet. The source then transmits the packet which takes L/r time, where L is the length of the packet and r is the transmission rate, the transmission delay is modelled by a time out for that processes. Once the simulation of transmission is complete the packet is put in the store signifying that the packet has reached the router.

1.3 Logging

In order to provide efficient debugging capabilities we created a logging utility that writes to a file the state of the queue during each event. This provides the user the ability to provide a full inspection to verify proper operation of the implemented queuing algorithm

1.4 Running Instructions

1.4.1 Installation

In order to use Simpy on the csa2.bu.edu server we must use virtualenv to be able to install Python modules without root privilege in a sandbox environment. We created an install script that will download, install and configure our test environment so the experiments can be run.

At the command line run the following:

```
./install.sh
```

If you are confronted with permission denied error you may need to change the permission on the file, at the terminal run:

```
chmod 755 install.sh
```

Grab a coffee or tea and when you return the installation process should be complete.

1.4.2 Running an experiment

It is important to note that the Python scripts **cannot be run directly**, they must utilize the virtualenv environment so the necessary modules will be imported. To make running experiments easier we have created bash scripts for each type of algorithm that includes the commands for running the code in the virtual environment. The main.py Python script will be executed in these bash scripts to run the experiment. This script has the following parameters:

-M [M] where [M] is an integer value specifying the total offered load, -x [X] where [X] is an integer value specifying the number of experiments to run, -n [N] where [N] is an integer value specifying how many packets to use in the experiment, -p is an optional flag indicating plots are to be generated and lastly the algorithm must be specified, one of –fifo, –rr, –drr, indicating FIFO, round robin and deficit round robin respectively.

An example of a typical command we used to run 10 experiments with 100,000 packets, total offered load of 1 for fifo algorithm and plot generation would be:

```
python main.py -x 10 -M 1 -n 100000 --fifo -p
```

As previously mentioned we made it easier to run in the virtual environment by creating individual scripts to run. To run an experiment simulating the FIFO algorithm run the following at the command line:

```
./exp_fifo.sh
```

To run an experiment simulating the round robin algorithm:

```
./exp_rr.sh
```

And deficit round robin algorithm:

```
./exp_drr.sh
```

If for any reason these commands fail for permission issues run chmod 755 and the file name was previously demonstrated.

1.5 Statistics Collection

All of our statistics are based on attributes stored on each packet. When a packet is created, it is given a 'length' attribute that indicates its length, in bits. During the simulation, times at which the packet 'arrives' at the router and 'departs' from the router are also stored; the router sets those time attributes for each packet object within its enqueuing and dequeuing methods. The packet is then capable of reporting its latency (queueing delay) by computing the difference between its departure time and arrival time.

After the simulation, the collected information is used to calculate latency and throughput. Average packet latency for a given source is computed as the sum of all of that source's packet's queueing delays, divided by the number of packets sent by that source. Throughput for a given source is computed as the sum of all of that source's packet's lengths, divided by the difference between the last departure time and the first arrival time.

2 Experimental Results

2.1 FIFO Router

2.2 RR Router

2.3 DRR Router

3 Discussion

Let q be the number of sources sending packets through the router, and b be the size of the router's buffer. c is a constant that is defined in our discussion below.

Algorithm	time complexity	space complexity
FIFO	$O(1)$	$b + O(1)$
RR	$O(1)$	$b + c + O(1)$
DRR	$O(1)$	$b + 2c + O(1)$

Note that for 'space complexity' we consider the total number of space required by the router throughout its lifetime, whereas for 'time complexity' we consider the cost of a single queueing or dequeuing operation.

All thee algorithms have constant time complexity. Note that a naive implementation of RR and DRR would have time complexity at least $O(log(q))$, in order to allow for finding the queue into which to add a new packet. However, both the RR and DRR can take advantage of hashing; they combine packets coming from sources which hash to the same value into one queue, thus capping the total number of queues at some constant value (2^c), and making it possible to find the queue corresponding to a given source in c time. Note also that stealing buffer space from the correct queue in constant time is possible due to McKenney's work on buffer stealing.

The space complexity differs for the three as follows: FIFO stores a single queue the size of the buffer. RR stores a buffer full of queues, with a map from sources (or source hashes) to the queues. DRR stores the same items that RR stores, as well as another map from sources (or source hashes) to their current deficit.

We think DRR is worth it. It provides a much stronger guarantee of fairness, while paying only a constant price in terms of space complexity. Routers are not terribly space constrained, and can easily handle the storage of another constant-sized map.